

# Ajax, Fetch API

---

Bản gốc: Phần 6 (Phần cuối): JSON và Ajax trong Javascript

## JSON

JSON (**J**ava**S**cript **O**bject **N**otation) là một chuỗi text được viết dưới dạng Javascript Object dùng để lưu trữ và trao đổi dữ liệu.

JSON thường được dùng để lưu trữ string, number, boolean, array, object, null.

Định dạng JSON có thể dễ dàng sử dụng ở các ngôn ngữ lập trình khác nhau. Ở Javascript bạn có thể thao tác với JSON thông qua `JSON.parse()` và `JSON.stringify()`

### Chuyển Object thành JSON

```
const myObj = {
  name: 'John',
  age: 30,
  cars: ['Ford', 'BMW', 'Fiat']
}
const myJSON = JSON.stringify(myObj)
console.log(myJSON) // '{"name":"John","age":30,"cars":["Ford","BMW","Fiat"]}'
```

### Chuyển JSON thành Object

```
// Chú ý là dùng dấu ' hoặc ` ngoài cùng thì JS mới cho phép
const myJSON = `{"name":"John","age":30,"cars":["Ford","BMW","Fiat"]}`
const myObj = JSON.parse(myJSON)
```

### Cú pháp JSON

- Data là cặp name/value
- Data được ngăn cách bởi dấu phẩy
- Ngoặc nhọn mô tả object
- Ngoặc vuông mô tả array
- JSON dấu nháy kép
- Trường name phải bọc trong nháy kép

### Value trong JSON phải là một trong những kiểu dữ liệu dưới đây

- string
- number
- object
- array

- boolean
- null

## Array và JSON

```
// Array
const arr = ['Ford', 'BMW', 'Fiat']
// JSON
const json = '[ "Ford", "BMW", "Fiat" ]'
```

Ngoài các giá trị được nêu bên trên thì các giá trị như **Function**, **undefined**, **NaN**, **Infinity**,... sẽ không được chuyển sang JSON nhé

```
var obj = {
  name: 'John',
  age: function () {
    return 30
  },
  city: 'New York'
}
var myJSON = JSON.stringify(obj)
console.log(myJSON) // '{"name":"John","city":"New York"}'
```

## AJAX

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX không phải là một ngôn ngữ lập trình

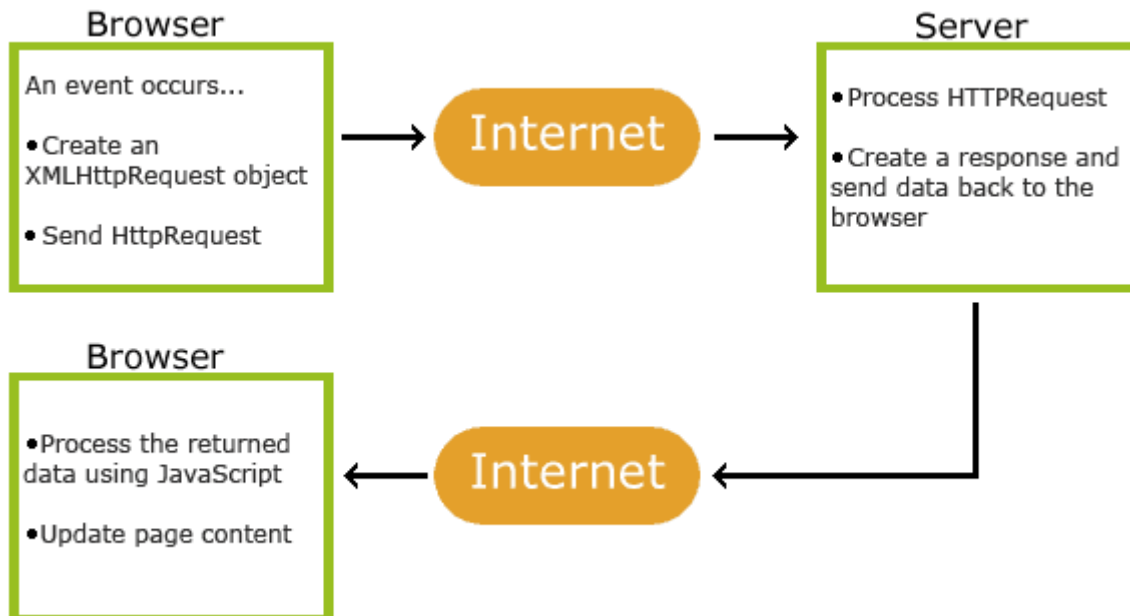
AJAX chỉ là sự kết hợp của

- XMLHttpRequest object có sẵn trên trình duyệt (dùng để gửi và nhận data từ web server)
- Javascript và HTML DOM (để hiển thị hoặc sử dụng data)

Cái tên AJAX dễ gây hiểu lầm là ứng dụng dùng AJAX sẽ sử dụng XML (một kiểu data như JSON nhưng phức tạp hơn) để gửi và nhận dữ liệu. Nhưng trên thực tế thì chúng ta chủ yếu dùng text hoặc JSON để trao đổi dữ liệu.


AJAX giúp chúng ta đọc dữ liệu từ server trả về, gửi dữ liệu lên server ở chế độ ngầm, cập nhật trang web mà không cần reload lại trang. Vì sự tiện lợi của AJAX mà chúng ta đã có React, Angular và Vue 😊

### Cách AJAX hoạt động



1. Một sự kiện xảy ra ở trang web (trang được load, một button được click)
2. Một XMLHttpRequest object được tạo bởi Javascript
3. XMLHttpRequest object gửi một request đến web server
4. Server xử lý request đó
5. Server gửi response về cho trang web
6. Response được đọc bởi Javascript
7. Thực hiện một số hành động trên trang web bằng Javascript (ví dụ như cập nhật lại trang)

## XMLHttpRequest

XMLHttpRequest (XHR) là một constructor function (mình đã giới thiệu về constructor function trong  bài này) dùng để giao tiếp với server. XHR cũng là một Web APIs nên nó chỉ có trên môi trường trình duyệt, không xuất hiện ở Node.js nha :mrgreen:

### Tạo một XMLHttpRequest object

```
var xhttp = new XMLHttpRequest()
```

### Một số phương thức của XMLHttpRequest object

Phương thức	Mô tả
<code>new XMLHttpRequest()</code>	Tạo mới XMLHttpRequest object
<code>abort()</code>	Hủy request hiện tại
<code>getAllResponseHeaders()</code>	Returns tất cả thông tin header
<code>getResponseHeader()</code>	Returns thông tin header chỉ định

Phương thức	Mô tả
open(method, url, async, user, psw)	Quy định request; method: Loại request (GET, POST, PUT, DELETE); url: đường dẫn đến server; async: true (bất đồng bộ) hoặc false (đồng bộ); user: tùy chọn username; psw: tùy chọn password
send(body)	Gửi body data đến server. body có thể là; Document, cần serialized trước khi gửi; Blob, BufferSource, FormData, URLSearchParams, or USVString object.null; Nếu không có giá trị nào cho body thì mặc định null được sử dụng.
setRequestHeader()	Thêm các giá trị vào trong header request

### Một số thuộc tính của XMLHttpRequest object

Thuộc tính	Mô tả
onreadystatechange	Xác định một function được gọi khi thuộc tính readyState thay đổi
readyState	Mô tả trạng thái của XMLHttpRequest. 0: request chưa được khởi tạo. 1: kết nối với server được thiết lập. 2: request được server tiếp nhận. 3: đang xử lý request. 4: request kết thúc và response đã sẵn sàng để dùng
responseText	Return về response như một string
responseXML	Return về response như một XML
status	Return về status của request. 200: "OK". 403: "Forbidden". 404: "Not Found". Xem danh sách đầy đủ tại <a href="#">Http Messages Reference</a>
statusText	Return về status dạng text (Ví dụ "OK" hoặc "Not Found")

### Ví dụ về cách dùng XHR để GET

```
function loadDoc() {
  var xhttp = new XMLHttpRequest()
  // function này sẽ chạy mỗi khi readyState thay đổi
  xhttp.onreadystatechange = function () {
    // Khi request thành công
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById('demo').innerHTML = this.responseText
    }
  }
  xhttp.open('GET', 'https://httpbin.org/get', true)
  xhttp.send()
}
loadDoc()
```

### Ví dụ về cách dùng XHR để POST

```
function send() {
  var xhttp = new XMLHttpRequest()
  // function này sẽ chạy mỗi khi readyState thay đổi
  xhttp.onreadystatechange = function () {
    // Khi request thành công
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById('demo').innerHTML = this.responseText
    }
  }
  xhttp.open('POST', 'https://httpbin.org/post', true)
  const body = { name: 'Du Thanh Duoc', age: 24, point: 100 }
  xhttp.send(JSON.stringify(body))
}
send()
```

Dùng `XMLHttpRequest` để xử lý AJAX được coi là cách "cổ xưa" nhất và rườm rà nhất. Ngày nay chúng ta có rất nhiều tùy chọn mạnh mẽ để xử lý AJAX. Mình có thể liệt kê ra

- Nếu dùng `Jquery` thì Jquery cung cấp cho chúng ta nhiều hàm gọi AJAX rất tiện lợi như `load`, `ajax`, `get`, `getJSON`
- Những trình duyệt ngày này cung cấp một API khác `XMLHttpRequest` đó là `Fetch API`, với nhiều tính năng nâng cao và cú pháp dễ sử dụng hơn `XMLHttpRequest`
- Phổ biến nhất phải kể đến `Axios`, một thư viện chuyên dụng cho việc xử lý AJAX cũng như gọi API. Cung cấp hàng tá tính năng hay, dùng được cho cả môi trường trình duyệt và Node (nếu trình duyệt nó sẽ dựa trên XHR, nếu là Node thì là `HTTP interface`)

## Fetch API

Fetch là một API đơn giản cung cấp cho chúng ta khả năng gửi và nhận request thông qua trình duyệt. Nếu như `XMLHttpRequest` dùng callback thì Fetch API dùng Promise, vì thế sẽ tiện lợi hơn khi thao tác và xử lý.

Một setup request đơn giản với fetch

```
fetch('http://example.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data))
```

Lưu ý là Promise return từ `fetch()` sẽ **không reject dựa trên HTTP error status** ngay cả khi HTTP 404 hoặc 500. Thay vì đó, nó sẽ resolve bình thường (với `ok` status là `false`), và nó chỉ sẽ reject khi mà mạng lỗi hoặc bất cứ điều gì ngăn cản request hoàn thành.

```
fetch('examples/example.json')
  .then(function (response) {
    if (!response.ok) {
      throw Error(response.statusText)
    }
  })
  // Do stuff with the response
```

```
})  
.catch(function (error) {  
  console.log('Looks like there was a problem: \n', error)  
})
```

## Đọc một response

```
fetch('examples/example.json')  
  .then(function (response) {  
    if (!response.ok) {  
      throw Error(response.statusText)  
    }  
    // Read the response as json.  
    return response.json()  
  })  
  .then(function (responseAsJson) {  
    // Do stuff with the JSON  
    console.log(responseAsJson)  
  })  
  .catch(function (error) {  
    console.log('Looks like there was a problem: \n', error)  
  })
```

Mặc định nếu không truyền tham số thứ 2 thì fetch sẽ dùng phương thức GET, còn muốn chỉ rõ phương thức như POST, PUT... thì phải truyền tham số thứ 2 vào.

### Ví dụ POST

```
// Example POST method implementation:  
async function postData(url = '', data = {}) {  
  // Default options are marked with *  
  const response = await fetch(url, {  
    method: 'POST', // *GET, POST, PUT, DELETE, etc.  
    mode: 'cors', // no-cors, *cors, same-origin  
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached  
    credentials: 'same-origin', // include, *same-origin, omit  
    headers: {  
      'Content-Type': 'application/json'  
      // 'Content-Type': 'application/x-www-form-urlencoded',  
    },  
    redirect: 'follow', // manual, *follow, error  
    referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-downgrade,  
    origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url  
    body: JSON.stringify(data) // body data type must match "Content-Type" header  
  })  
  return response.json() // parses JSON response into native JavaScript objects  
}  
postData('https://example.com/answer', { answer: 42 }).then((data) => {
```

```
    console.log(data) // JSON data parsed by `data.json()` call  
  })
```